CS1026 – Assignment #3
Country Classes

Due: Thursday, June 10, 11:55pm on OWL
Weight: 16%

Learning Outcomes

By completing this assignment, you will gain skills relating to:

- strings and text files;
- writing and using your own classes;
- using complex data structures (i.e., lists, sets, dictionaries);
- testing code and developing test cases;
- adhering to specifications.

Task

Data files are very common in many disciplines. They form the input for a variety of different kinds of processing, analysis, summaries, etc. An associated problem is to maintain these data files – to add information, update or correct information. Manually updating data files can be tedious and can lead to errors. A common approach is to use a program – an update program that takes a data file and a file of updates and then creates a new version of the data file with the updates applied.

In this assignment you will create a complete program that will update an existing data file containing information about countries. The tasks are outlined below.

Functional Requirements/Specifications

Task 1

Implement a class **Country** that holds the information about a single country. The file that holds this class will be named `country.py` and will contain the following:

1. **Instance Variables**
   - `name`: the name of the country (string)
   - `population`: the population of the country (string)
   - `area`: the area of the country (string)
   - `continent`: the name of the continent to which the country belongs (string)

   Note: the instance variables *population* and *area* can be left as strings for this assignment, as there are no computations involving either. The update program just needs to update these values and save them to a file.

2. **Methods**
   - Constructor, `__init__ (self, name, pop, area, continent)`
   - Getter Methods: `get_name`, `get_population`, `get_area`, `get_continent`
   - Setter Methods: `set_population`, `set_area`, `set_continent`
   - `def __repr__ (self):` generates a string representation for class objects

   Example of output from `__repr__`

```
Name (pop: population value, size: area value) in Continent

e.g., Nigeria (pop: 11723456, size: 324935) in Africa
```

Test all of your classes and methods before moving to the next section. Feel free to create other helper methods if necessary.

Task 2

Implement a class called `CountryCatalogue`; name the file `catalogue.py`. This class will use a file to build the data structures to hold the information about countries. The file `data.txt` contains information about a number of countries.

1. **Instance Variables**
   - `country_cat`: this is a collection of countries. Each member is an object of the class `Country`. The collection could be a set, dictionary or list. **It is important that the collection stores objects of type `Country`.**

   - **You may add other instance variables as you see fit.**

   An example of the `data.txt` file is:

   ```
   Country|Continent|Population|Area
   Brazil|South America|213,952,606|8,515,767
   Canada|North America|38,048,738|9,984,670
   China|Asia|1,444,390,177|9,596,961
   Colombia||50,372,424|1,141,748
   Egypt|Africa|101,576,517|
   ```

   Notice that:
   - There is a header line: `Country|Continent|Population|Area`. This tells you how the data is formatted in the file.

   - The file consists of *records* for each country – one per line, with the information as described in the header line.

   - The different *fields* of each *record* are separated by vertical bars: `|`.

   - A *field* (except for country name) may be missing (see Columbia and Egypt), but the record will still have the three vertical bars.

2. **Methods**
   - Constructor, `__init__ (self, country_file)`: The constructor method will take a single parameter (besides `self`) that is the name of the file that contains the information about each country. The constructor will use the data in the file to construct the data structure `country_cat`. A sample data file has been provided: `data.txt`, `data2.txt` and `data3.txt`. **[Note: all files have headers, and these files are meant for you to test your program; these may NOT be the same files used to evaluate your solution.]**

- Setter Methods: `set_country_population`, `set_country_area`, `set_country_continent`

- `find_country(self, country)`: Given a country object (that is, an object of the `Country` class), this method checks to see if that country object is in `country_cat`. If it is, it just returns the country object, but if it is not, it returns the null object: `None`.

- `add_country(self, country_name, pop, area, cont)`: Given the name, population, area and continent of a country, this method adds a new country to `country_cat`; it should only be added if the country does not already exist in `country_cat`. The method should return `True` if the operation is successful (country successfully added), or `False` if it is not (e.g., the country you entered already exists in the catalogue).

- `print_country_catalogue(self)`: This method prints a list of the countries and their information to the screen; it should make use of the `repr` method of the `Country` class.

- `save_country_catalogue(self, fname)`: This method will enable all the country information in the catalogue to be saved to a file. The method takes as a parameter the name of the file. All the countries should be sorted alphabetically by name (A – Z) prior to saving. If the operation is successful, the method should return the **number of items** written; otherwise, it should return a value of `-1`. **The file should appear in the exact same format as the original file, namely:**

```
Country|Continent|Population|Area
Brazil|South America|213,952,606|8,515,767
Canada|North America|38,048,738|9,984,670
China|Asia|1,444,390,177|9,596,961
Colombia|South America|50,372,424|1,141,748
Egypt||101,576,517|1,010,408
Indonesia||260,581,345|
```

- **You may add other methods as you see fit.**

Notice that:

- There should be no spaces between the fields and vertical bars.

- If a field has no value, then nothing should be printed (e.g., see Egypt and Indonesia with no continent value and Indonesia with no area given).

Test all of your classes and methods before moving to the next section. Feel free to create other helper methods if necessary.

Task 3

Implement a Python module, called `process_updates.py`, that has a function `process_updates(country_fname, update_fname)`. This function takes two parameters: the first parameter will be the name of a file containing country data (e.g., `data.txt`). The second file will contain the *name of a file containing updates*. Each record in the update file specifies updates for a single country. **The format of a record in the update file is as follows:**

```
Country;update1;update2;update3
```

Where an 'update' is of the form "L=value", where L can be **POP**, for population, **AREA**, for area, and **CON**, for continent. Updates are separated by semicolons and a line can have 0, 1, 2 or 3 updates; **any updates after the first 3 are simply ignored, and you need not worry about their values**.

Valid values in an update are as follows:

- For **Population** and **Area**, a valid value is a string of digits with commas separating groups of 3. Examples of valid values are "123,456", "1,333,456", "23,333,444". Examples of invalid values are "123456", "123 345 555", "123,344.44".

- For **Continent**, a valid value is one of the continents: "Africa", "Antarctica", "Asia", "Australia", "Europe", "North America" and "South America". The string of letters can be uppercase or lowercase, and there may be more than one space between the "North" and "America" and the "South" and the "America". **Once a valid**

continent is found, the first letter of the name or parts should be capitalized, and there **should only be 1 space** between the "North" and the "America" and the "South" and the "America". For example, "asia" should be stored as "Asia", "aFRICA" should be stored as "Africa", "north   America" should be stored as "North America".

An example of an update file is:

```
Brazil;POP=193,364,111;AREA=8,511,966
Indonesia;POP=260,581,345
Italy;POP=59,801,999
Japan;POP=127,380,555;AREA=377,800
Sweden;POP=9,995,345;AREA=450,295;CON=Europe
Vietnam;AREA=331,310;POP=95,541,921;CON=Asia
France;POP = 64,668,129;POP=65,668,150;AREA=541,666;CON=Europe
```

Notice that:

- The updates can be in different orders (e.g., the update for Vietnam is area first, then population).

- If a country is specified and that country is **NOT** already in the catalogue, **then that country and its attributes (at least the ones specified; not all have to be specified) should be added to the catalogue (again, any attributes beyond the first 3 are to be ignored).**

- There can be spaces before and/or after either the semicolons or the equal signs; these spaces can be ignored.

- There can also be spaces in other places, such as in country names or within numbers. These represent errors and could cause problems in processing. **You should treat these errors (do not try to fix them), and they should be handled as exceptions (see below).**

- In the updates to **France**, for example, there are four updates; the last update is simply ignored, and your program should only consider the first three updates. As well, the first three updates to France contain two updates for population. These are applied in order. This would mean that the update **POP=64,668,129** is applied, and then the update **POP=64,668,150** is applied, essentially overwriting the first update.

How `process_updates()` works:
As noted above, the function `process_updates()` gets two files as parameters. The first parameter is the name of the file with the country data, and the second parameter is the name of the file with the update data.

The country file should be processed first. The function `process_updates()` should ensure that it exists. If it does exist, it should then proceed to process the data file using the class `CountryCatalogue`. If the country file does not exist, then `process_updates()` should give the user the option to quit or not (prompts for "Y"(yes) or "N"(no)). If the user does not wish to quit (i.e., they respond with "N"), then the function should prompt the user for the name of a new file. If the user wishes to quite (i.e., responds with a "Y", or anything other than "N"), then the method should exit and return `False`. The method should continue to prompt for file names until a valid file is found or the user quits.

If the user chooses to quit with no country file processed, then the function `process_updates()` should **NOT** try to process the updates file. Instead, it should write to the file `output.txt` the message `"Update Unsuccessful\n"` – just this single line. It should then exit and return `False`.

If the function `process_updates()` **has successfully processed the country file**, then it should proceed to process the file of updates. It should prompt the user in a similar manner to how the country file was input (i.e., using a loop to continuously prompt the user until a file is found that exists or until the user quits).

If the user quits without an update file being selected, then the function `process_updates()` should write to the file `output.txt` the message `"Update Unsuccessful\n"` – just this single line. It should then exit and return `False`.

If an update file is found, then the function process_updates() should process the updates (i.e., update the information about the countries and then output the new updated list of countries in alphabetical order to the file `output.txt`. The output should use the method `save_country_catalogue(self, fname)` from the `CountryCatalogue` (see Task 2). It should then exit and return `True`.

In processing the updates from the update file, the updates may have invalid values (see above**). If an update has an invalid value, then the entire line should be skipped, and none of the updates should be processed.** It is difficult to anticipate all possible problems with the updates, and some may cause your program to fail – that is, some invalid input MAY cause exceptions when processing the line. Your program should be designed to catch any of these exceptions and then print a message and skip all processing of that line.

The Python program `main.py` file is provided as the interface to your program. It has been designed to prompt for the names of a country file and a file of updates, and then it calls the function `process_updates()`; make sure that you use the names of functions and methods as described above, or your program may not function. You may use `main.py` to test your program in other ways. As mentioned above, three data sets of countries have been provided (`data.txt`, `data2.txt` and `data3.txt`), as well as three different files of updates (`upd.txt`, `upd2.txt` and `upd3.txt`). **These files can be found in Resources > Assignment 3 on OWL**. You may use these to test your program. **NOTE: while `main.py` will test some aspects of your program, it does not do a complete and thorough testing – this is left up to you. A program similar to, but different than, `main.py` will be used to test your program – it will include other tests.**

The following may be helpful in sorting data structures:
http://pythoncentral.io/how-to-sort-a-list-tuple-or-object-with-sorted-in-python/
http://pythoncentral.io/how-to-sort-python-dictionaries-by-key-or-value/

Non-Functional Requirements/Specifications

1. Include brief **comments** in your code identifying yourself (i.e., include your name in a comment at the beginning of your program), describing the program, and describing key portions of the code.

2. Assignments are to be done individually and **must be your own work**. Software may be used to detect academic dishonesty (cheating).

3. Use Python coding conventions and good programming techniques. Examples include:

- meaningful variable names;
- conventions for naming variables and constants (you can use underscores or camel case, but be consistent);
- readability – indentation, whitespace, consistency.

4. The name of the files you submit **MUST** be as follows:
   - `country.py` containing the class `Country`;
   - `catalogue.py` containing the class `CountryCatalogue`;
   - `process_updates.py` containing the function `process_updates()`.

5. You will attach the files in OWL. **DO NOT** compress/zip them. **DO NOT** just enter code into the text submission area on OWL.

6. Make sure to use **Python 3.9** as the interpreter; failure to do so may result in the testing program failing.

Marking the Assignment

1. Your program will be executed by an automated testing program. This testing program assumes that:

   - The modules are named `main.py`, `process_updates.py`, `catalogue.py` and `country.py`.

   - That you are using Python 3.8 or 3.9.

   - That you submitted via OWL correctly (i.e., **NOT** compressed).

2. The TAs will be looking at the following things when grading your assignment:

   - Are the modules, classes and functions named properly for testing?

   - Is there a program `process_updates.py` which has a function `process_updates()`?

   - Does the program work with the test program, `main.py`?

- Is there an effective use of functions and methods?
- Is the output according to specifications?
- Is the code commented clearly? Does it contain meaningful names?

The TAs will also be checking to ensure that things were not hardcoded and that your program actually uses the techniques learned in this course.